Teaching HW/SW Co-Design With a Public Key Cryptography Application

Leif Uhsadel, Markus Ullrich, Amitabh Das, Duško Karaklajić, Josep Balasch, Ingrid Verbauwhede and Wim Dehaene, Member IEEE

***Abstract*—This paper describes a lab session-based course on hardware/software (HW/SW) co-design. Real problems often need to combine the speed of a HW solution with the flexibility of a SW solution. The goals of this course are to show that there are many alternative solutions in the design space, and to teach the fundamental concepts of HW/SW co-design. The sample application for the course project is a basic public key (RSA) application. This application is attractive for pedagogic purposes, because its complex arithmetic and large word lengths make it difficult to realize in SW on an embedded micro-controller. But the alternative of a pure ASIC (application-specific integrated circuit) application is also not a satisfactory solution, as this lacks the flexibility to support multiple public key applications. The project follows a step-wise approach, with assignments that build on each other. Students are required to make their own decisions as to the partitioning between HW and SW, the interface design, and the optimizations goals. Besides imparting hard skills in HW design and embedded SW design, the course inculcates several soft skills in particular, decision making, presentation skills, teamwork and design creativity - generally overlooked in engineering.**

***Index Terms*—HW/SW co-design, public key cryptography, Montgomery, RSA, 8051 microcontroller**

I. Introduction

THE recently revised M.Sc. program in Electrical Engineering at KU Leuven, Belgium, offers students two specialization tracks: *Electronics and Integrated Circuits (EIC)* and *Embedded Systems and Multimedia (ESM).* The course *Design of Digital Platforms* is part of the common core education modules taught during the first semester, and it serves to bridge the two tracks. Students who follow the EIC track need to understand the various applications that will run on the platforms so that they can build more efficient and stable processors. Students who follow the ESM track need to understand digital platforms so that they know what performance can be expected from them. In its earlier format, the M.Sc program had no bridging course, and the tracks were more independent.

The goal of this Design of Digital Platforms course is to introduce students to the world of Hardware/Software (HW/SW) co-design of electronic systems [1]. The idea behind HW/SW co-design is to address the development of HW and SW components jointly when designing an application. A large design space offers many choices for design decisions and optimizations; at the same time these choices influence trade-offs between area, execution time, power, energy, flexibility, and design effort. In this course, students gain understanding of this large design space through practical experience. The focus of the course project is to optimize the implementation in terms of three of these design dimensions (area, execution time and flexibility), as will be explained below. For future offerings of the course other optimization goals, such as energy consumption for battery-operated devices, would be valid options. The course guides students step-by-step through the SW and HW design, and then combines the two approaches in a final stage. By first designing HW and SW separately, students experience the limitations and advantages of both. This prepares them well for the co-design phase of the project, where the focus shifts to higher-level design decisions, such as time-performance trade-off management, HW/SW boundary selection, and interface design.

The trade-offs in system design are best taught by complementing theoretical classes with hands-on experience. This approach has also been followed in other related works. For instance, [2] presents a co-design course applying symmetric key ciphers. In [3] a helicopter-like robot motion control is implemented, while [4] discusses co-design as an emerging discipline in education. For the course described here, a cryptographic system was chosen as the application, for several reasons: first, efficient implementation of cryptography is a common problem in industry; second, the modular nature of cryptographic algorithms allows a nice division of the tasks into multiple teaching sessions; and third, the computational complexity of public key cryptography becomes a challenge when targeting embedded devices.

Cryptographic systems are used to enable security in a wide range of IT-related fields. RSA [5] is a standardized and popular public key cryptographic system for key exchange and digital signatures. Its main computation is modular exponentiation using long integer operands, typically larger than 1,024 bits, making computational demand one of the drawbacks of public key cryptography. Implementing public key cryptography on micro-controllers is known to be a difficult task, but one that is nevertheless needed for smart card applications such as banking cards or e-IDs. SW-only solutions are possible on some micro-controllers, but require a huge amount of optimization effort while consuming most of the micro-controllers resources. HW-only implementations, on the other hand, are fast, but consume more silicon area and are not as flexible as SW solutions since they cannot be easily updated once deployed. HW/SW co-design thus appears to be a natural solution, by combining the speed of HW with the flexibility of SW.

All authors except Prof. Dehaene are with KU Leuven, ESAT/SCD-COSIC and iMinds; Prof. Dehaene is with KU Leuven, ESAT/MICAS and iMinds. Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium

The goal of the project-oriented course is to implement an RSA cryptographic system using an 8-bit micro-controller with an attached co-processor. The contribution of this paper is to provide a thorough education-oriented description of the project-based HW/SW co-design course, including discussions of educational goals, results, grading, and plagiarism detection.

The remainder of the paper is organized as follows. The course's teaching goals are explained in Section II and the teaching approach is presented in Section III. In Section IV, a detailed description of the course is given. Section V covers the software assignments, and Section VI focuses on the hardware assignments. The final task of a co-design implementation is presented in Section VII. The results of the course, covering implementation results and grading methodology, are given in Section VIII. The paper is concluded in Section IX.

II. Educational goals

The educational goals of this course are fourfold: digital platform understanding, co-design experience, hard skills and soft skills.

Digital platform understanding: Understanding the role of the digital platform as the link between an application and its realization is a key design aspect. When implementing next generation embedded systems, the choice between HW (ASIC[1] design) and SW (coding for a general purpose processor) is not binary. Indeed, in reality there is a large design space between the two extremes of pure HW or pure SW. Thus for the future embedded systems designer, the correct selection and understanding of the target platform is of extreme importance. Similarly, the future integrated circuits designer needs to understand the complexity of the applications that will run on the digital platform he or she will design.

Co-design experience: The many design choices available in HW/SW co-design will affect the final result. These choices often represent a trade-off, and students will learn that there is no single solution. Part of the teaching goal is to encourage students to focus their optimization in the triangle: execution time, hardware area and flexibility, Fig. 1. Flexibility is the systems ability to adapt with little overhead cost in terms of both performance and cost. It is thus measured indirectly, by analyzing the performance and cost of a related application on the same HW/SW platform. Cost can be expressed by hardware size and/or energy or power consumption, while execution time is measured by throughput or clock cycles. In the course, cost is measured by hardware size and time by throughput. SW is easier to update than HW, and architecture features should support multiple applications without increasing hardware cost. A pure software solution will be low cost (as no additional hardware is required) and will offer the most flexibility, but will be the slowest. A large co-processor or full custom design will require the most additional space for hardware and will offer no flexibility, but will be the fastest solution. Students have to address this trade-off between

[1]ASIC: application-specific integrated circuit

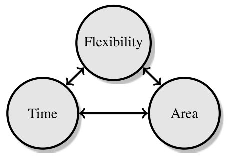

Fig. 1. HW/SW co-design trade-off

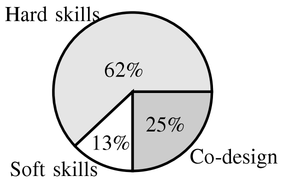

Fig. 2. Distribution tasks (based on course schedule)

performance, cost and flexibility. Every decision take shifts the final solution towards a different trade-off position.

Hard skills: Hard skills comprise the technical skills required to carry out the co-design project; these include knowledge of several programming languages for HW and SW design, of co-design simulation, of the corresponding development tools. Understanding of the platform characteristics and the implemented algorithms is also required. These hard skills, particularly experience in one or several programming languages, can be either a course prerequisite or, alternatively, a teaching goal.

Soft skills: Soft skills or non-technical skills addressed in this course are decision making, oral presentation and discussion, writing a report, organizing workload to meet deadlines, and working in a team including sharing workload. Soft skills are often ignored in technical studies. The co-design project offers several opportunities to integrate soft skills. The students choose their optimization goal, and this decision has to be explained via a written report and oral presentations in front of the class, professor and TAs. Students also have to learn to make a plan, and keep to schedule. For the SW tasks, since limited time is given to finishing the task, they have to decide when to stop optimizing the code. Additionally, students work in pairs; each pair must discuss problems, decide how to allocate tasks and share code, and set intermediate milestones for each assignment.

III. Approach

This section explains the approach taken to achieve the educational goals described in Section II.

Theory. A limited set of traditional lectures explain various design methodologies and design steps, covering the theoretical aspects of data-flow and control-flow analysis, and the impact of memory management and transformations. These lectures also cover design for high throughput and/or low power/low energy. Finally, students are exposed to a wide range of digital platforms for embedded devices, among them ASIC, ASIP, FPGA, domain-specific processors, embedded micro-controllers. Note that this paper does not cover the theory part and is focusing on the course project.

Co-design experience. This is inculcated in a two-phase project-based approach. In the first phase, students carry out several assignments in which hardware and software are treated separately. In this phase, which includes implementation and optimization tasks, students gain experience on the platforms, and of the various properties of hardware and software. In particular, they learn the limits of software and thus the need for a co-processor. In the second phase, students have to build a co-design, using the modules developed during the first phase as a starting point. One of the critical remaining tasks at this point is to select the HW/SW boundary, that is, the design of the interface between hardware and software along with a concept for data flow and control flow between the hardware and the software.

Hard skills. In addition to the lab sessions, extra exercises remedy students' lack of background knowledge or experience in the programming languages used during the course. Should students have a strong background in this field, the course could be given in less time, or with more ambitious optimization goals, or with more complex applications.

Soft skills. Students work in pairs throughout the semester. Teamwork being one of the teaching goals of the course, a more realistic teamwork scenario, especially with respect to post-graduation employment, is achieved by pairing students of different academic backgrounds. Typically the pair will combine students with integrated circuit background and embedded systems background, and with a differing Bachelor's degree. Most students will have taken a high-level programming language course (such as Java) at the Bachelor's level, but may not have experience in hardware description languages.

Application. Commonly encountered in industry, co-design of public-key systems on embedded microcontrollers is also well suited for education. First, public-key algorithms allow for a modular design with a step-by-step bottom-up approach. This makes it possible to split the project into the required assignment modules, which can be reused in subsequent assignments. (See Section IV for a description of all modules.) The course uses the 8051, an old, but commonly-used, micro-controller. It is widely employed in the industry and comes with free development tool chains. Further, its limited resources leave students in no doubt that additional hardware is required to carry out public-key cryptography on this platform.

IV. Course Description

This section describes the structure and content of the course project.

Bottom-up design approach. Public key algorithms can be split into three layers of abstraction, Fig. 3. For this project the cryptographic algorithm RSA was chosen. RSA is a so-called asymmetric key algorithm, which means that the key consists of two parts: the public and the private. The principle can be best compared with a padlock; anyone can close an open padlock but then only the person with the key can open it. Data encrypted with the readily available public key can only be decrypted by the owner of the private key. The RSA algorithm is based on modular exponentiation. The cipher text, for example, is calculated with the formula $c = m^e \pmod{n}$, where m is the message and e and n are part of the public key. Further details can be found in [5].

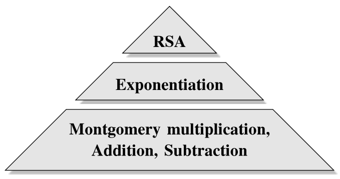

Fig. 3. Levels of abstraction of RSA

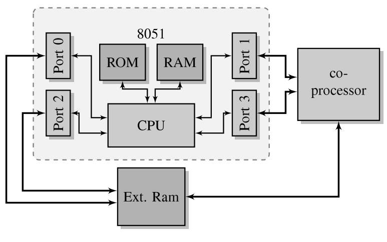

Fig. 4. Architecture overview

The next lower layer contains the arithmetic operations required such as modular long integer exponentiation. A schoolbook square and multiply algorithm is used here; for example, $x^{5_{10}} = x^{101_2} = ((((1)^2 \cdot x)^2)^2) \cdot x$, see [6]. The bottom layer comprises the required modular multiplication, modular addition and subtraction. The multiplication is the most important algorithm as it is responsible for most of the execution time. For this project, the Montgomery multiplication [7], a common approach for modular multiplications avoiding costly inversion or division-based reductions, was chosen.

Architecture. The 8051 micro-controller [8] was chosen as the target platform for software development. The 8051 is a lightweight 8-bit micro-controller, commonly used in low cost applications, whose architecture is illustrated in Fig. 4. It has four 8-bit parallel ports to communicate with peripherals. Two of these ports can also be used to extend the limited internal RAM; the basic model only supports 128 bytes, with additional external RAM. The external RAM can also be accessed by co-processors.

Timeline. The project timeline is shown in Fig. 5. In addition to the theory lectures that run in parallel with the project, there are three other types of lectures. First, exercise sessions in the computer classes give training in practical skills, such as introduction to C, GEZEL [1], and first steps with the toolchain. A second set of sessions serve to explain and implement the building blocks of the step-by-step approach. Third, at the end of course there are several free sessions during which teaching assistants (TAs) are available to answer questions and provide guidance.

The timeline shows that after a short introduction to C, the software assignments start right away. The practical sessions on the GEZEL hardware description language run in parallel, followed by an exercise introducing interface techniques and simulation. By week six, these exercises and assignments finish with the first milestone, the Montgomery multiplication in software, and students give an oral presentation. Subsequently, in the second milestone, they rebuild a Montgomery multiplier in HW, for which they can reuse code from the initial exercises. When the co-processor building blocks are ready,

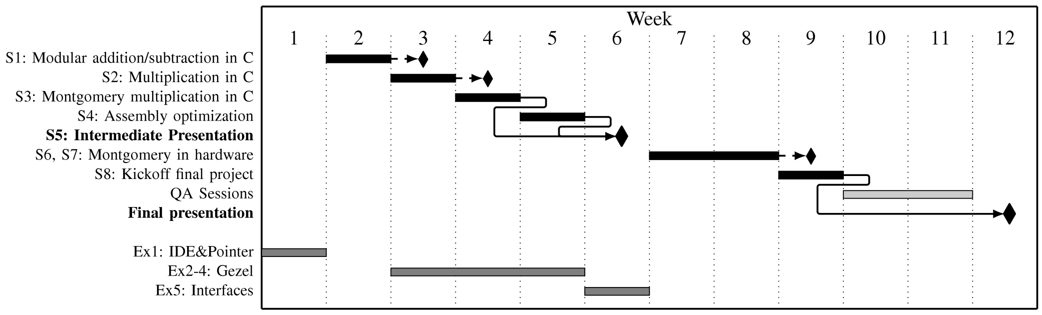

Fig. 5. Course schedule: ■ represents project sessions, ▨ stands for exercise session and □ indicates free working sessions. Deadlines are marked with ♦.

they then create an interface. In the final sessions, students are encouraged to profile their code, find bottlenecks and improve their interface design and apply further optimizations according to their individual design goals.

Toolchain. For software development students are provided with the *s*mall *d*evice *C c*omplier SDCC [9] and the *i*ntegrated *d*evelopment *e*nvironment MCU 8051 IDE [10]. For hardware development GEZEL is used [1]. GEZEL is a high level hardware development tool-chain comprising a compiler, which can be translated to VHDL for further synthesis, and a cycle-accurate simulator. GEZEL comes with co-simulation support for a set of micro-controllers, including the 8051, and allows for easy and quick co-simulation of hardware *and* software. All the course tools are free to use, and bugs found by students are always reported to the tool developers.

V. Software Assignment

The software assignment covers a total of five sessions at the start of the course, see Fig. 5. The goal of this step is to highlight the difficulty of efficiently implementing very large word length operations (1024 bit squaring in RSA) in resource constrained SW platforms such as the 8-bit 8051. As a side effect, the students also learn the negative impact of using a high-level programming language, such as C, instead of a low-level language, such as assembly, when aiming for efficient arithmetic implementations.

Starting from sample code, students are expected to implement all operations required by the cryptographic algorithm in the lower multi-precision and modular arithmetic layers. These include addition, subtraction, and multiplication.

Students are given reference algorithms for software implementations from various sources [6], [11] to establish a common basis from which to compare the implementations.

In the last session, dealing with software optimization, students are asked to improve their Montgomery modular multiplication. The compilation process from C to 8051 machine code leaves plenty of room for optimization. Students are asked to identify the bottlenecks in their current implementations and rewrite them in assembly language or using inline assembly. Typical examples of bottlenecks when implementing arithmetic operations are unnecessary accesses to memory, storage of intermediate results in external memory, inefficient loop control routines, and handling of carry bits.

Feedback to students: For each of the two milestones, (see Section IV-Timeline , an online spreadsheet is set up. Students are asked to enter their results in the spreadsheet as soon as they have a working version, and to update this as their results improve. The spreadsheets serve two purposes: to provide students with continuous feedback on the performance of their implementation, and to create a competitive environment that encourages students to apply further optimizations and find more creative solutions.

VI. Hardware Assignments

The hardware part of the course consists of five sessions, covering RTL modeling and teaching the *parallel* nature of hardware. This is in contrast to software, which runs *sequentially*.

Two important principles in hardware description are introduced: 1) *cloning*, which allows multiple instances of a single data path to be created, and 2) *combining*, which allows complex hierarchical designs to be built from the smaller blocks. For instance, an N-bit adder may be later reused to implement the Montgomery multiplier [12]. At this point, students can perceive the advantages of hardware modeling. By choosing a 1-bit digit-size, the Montgomery multiplication algorithm is simplified and made easier to implement in comparison the software assignments, where the digit size matches the word size of the processor (1 byte). During this phase, students are advised to minimize the area consumption of their designs by reducing the number of registers and by implementing a bit-serial version of the algorithm.

The VHDL code obtained (generated from the Gezel code using the integrated converter tool) is synthesized for FPGA using the Xilinx ISE environment. The target platform, Xilinx Virtex 4 XC4VFX60 [13], was chosen because it is has enough resources to implement even the very inefficient and area-consuming designs that students are likely to produce, due to their lack of experience. Students are asked to report the following performance results:

- Area occupied by the FPGA, in percent.
- Number of occupied Lookup Tables (LUTs).
- Number of flip-flops in the design.
- Maximum clock frequency and the critical path.

VII. Co-design

The co-design starts with an introduction to the existing HW/SW interfaces supported by the 8051. In the final class assignment, students combine their HW and SW modules and then optimize their design during the Q/A sessions. This assignment resembles the second milestone. Again students make a performance comparison using a second shared spreadsheet as in the software part, Section V. Also, students must defend their design in a presentation and hand in a report.

Since the micro-controller and the co-processor run at different clock speeds, handshaking through acknowledgement signals must be established, along with a strategy for efficient control flow and data flow. The HW/SW interface can easily become the bottleneck of the co-design, as will be seen from the results.

As with the first milestone, students have to defend their design in front of the class, report their performance measurements in a spreadsheet shared with the class, and write a report that covers:

- the technical results such as execution time and size of hardware
- the design methodology and test approach
- the major design decisions and optimizations

VIII. Results

A. Technical

Fig. 6 shows the students' design solutions in terms of area, time, and flexibility. Good overall designs are closer to the origin of the scatter plot. While the distance to the origin serves as a measure of the quality of the design, it does does not take flexibility into account.

Flexibility is difficult to measure, and solutions are sorted into three sets. Basically, the more of the system remains in software, the more flexible it is considered. Flexible designs (A) are considered to be those where the exponentiation algorithm is implemented in SW on the micro-controller, while the computationally costly arithmetic operations are performed in HW. Less flexible approaches implementing the entire protocol in HW are classified as (B), while other approaches are summarized as (C).

Since students in this particular course have very little coding experience it can be difficult to see the trade-offs , in terms of area or execution time when comparing teams. Teams with stronger coding skills often achieve a system that is faster and smaller. But while this trade-off cannot be shown on the graph, student feedback indicates that they do experience this trade-off during their *own optimization process*.

The graphs further show that the HW/SW boundary does not necessarily impact speed. Apart from the aforementioned impact of students low experience of coding, it can be seen that a well-designed interface can reduce the overhead of data and control flow between HW and SW to an almost negligible level, see also [14]. Certainly the pairs with good results for optimizing runtime and hardware size are also likely to achieve better interface design.

It can be seen that fast HW designs (category B) typically pay a price in area. When (A)-designs are slow, this is a result of a bottleneck in the interface and the SW part. Most common examples are unnecessary data flow, inefficient control flow, or slow exponent scanning in SW.

B. Evaluation

This section describes the approach taken to grading, and discusses the definition and detection of plagiarism in the course.

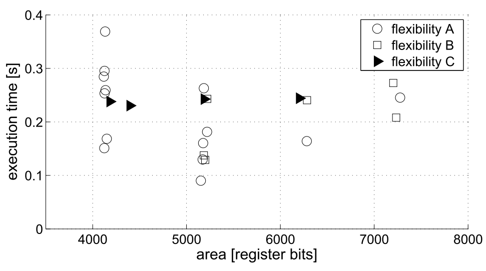

Fig. 6. Time-area-flexibility trade-off of student solutions

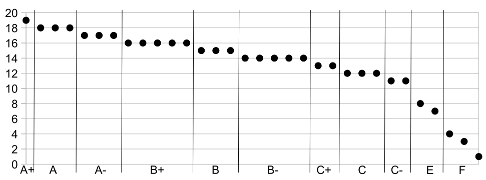

Fig. 7. Distribution of grades

1) Grading: Assessment data for grading is generated twice during the project: after the first and second milestone. The source code resulting from assignments prior to the milestones is completely reused, hence an evaluation of the milestone-assignments evaluates the whole course. For simplicity reason we will discuss only the combined criteria for the final grade.

Given that a HW/SW co-design is a tradeoff between area, speed and flexibility, all three aspects are important for grading, but while area and speed can easily be quantified, that is not true for flexibility. Also the project uses only basic algorithms and just gives an introduction to a simple interface. Both leave a lot of optimization potential open. Creative strategies in the interface design and creative optimizations in hardware and software are also taken into account. It could be said that these last two criteria are already covered in addressing speed, area and flexibility. But it can happen that solid straightforward solutions achieve similar or better performance than a creative solution with several good optimizations but which has a single overlooked bottleneck. Giving extra credit for creativity helps in balancing the grades.

The basic final grade is directly calculated from the area and speed of the final project. This base grade is then adjusted by a broad range of adjustment criteria, a set of positive and negative points covering the quality of the report, presentations skills, quality of code, flexibility of the design, interface between hardware and software, and creative optimizations. Clearly there can be a significant adjustment of the base grade in either direction, but not to the extent that a poor performance design can have a top score or that a high performance design will fail the course.

The pairs grades are given as points according to the local university system; Fig. 7 shows these as U.S.-type grades. These grades show that many students pick up the design methodology quite well, with only a few failing the course or getting a bad overall score. Most achieve quite solid results and several achieve the top score.

2) Plagiarism: Plagiarism is a general problem, and especially in a course which requires students to write code. Staff

TABLE I
MEASURABLE LEARNING PARAMETERS

Measurable Learning Parameter	Very effective	Effective	Not so effective	Ineffective
Improving the programming skills of the students in Embedded C	7	33	9	1
Improving the technical skills of the students in a hardware description language (Gezel)	15	28	6	1
Improving the coding skills of the students in Assembly language	7	18	19	6
Improving the students' understanding of architecture of a co-design project	6	33	9	2
Guidance of the Teaching Assistants	28	18	4	0
Structure of the course	20	14	15	1

and students do not always have the same definition of plagiarism, [15]. In this course students are encouraged to discuss problems with each other and help one another to develop solutions and share ideas. However, copying code is strictly forbidden. This is detected by identifying suspect candidates and then performing manual code analysis supported with diff tools. Suspect candidates are those who have exact matches in runtime or hardware size, very similar designs, or who made slow progress during the seminars followed by outstanding results.

C. Teaching Goals

Co-design methodology, hard skill, and soft skill teaching goals are evaluated. Feedback is collected in the form of a questionnaire after the course. The questionnaire is anonymous and optional. In the year 2011/12, 21 of 96 students answered it, while in the following year responses were received from 29 out of 98 students. Some of the main results are summarized in Table I. The students were quite satisfied with the growth that they could see in their software programming and technical skills in Gezel, with the majority answering in the affirmative. Most of the students also improved their understanding of the architecture of co-design projects. The guidance provided by the TAs and the overall structure of the course was also found to be quite effective.

When asked if they would use technical concepts from the course in the future, fourteen answered they definitely would, twenty-five answered probably, eight were undecided, and only three did not answer. The close contact of TAs with the students during the course also gives the authors a positive impression of the learning effect of the students. Given this, and the many creative solutions seen from code analysis, the *co-design* concept can be considered successful. Code analysis from the beginning and end of the course reveals that the *technical skills* of the students, especially coding, improves a lot. The students rate working in a team as being congenial, which matches the impression of those supervising the classes. The other *soft skills*, oral presentation and written report show variable results, underlining the importance of improving the integration of soft skills in technical studies.

IX. CONCLUSION

HW/SW co-design experience helps students learn to overcome the individual shortcomings of hardware and software and combine them to achieve fast, small and flexible designs. By implementing a complex cryptographic algorithm, students become familiar with advanced long number arithmetic techniques, widely employed in most signal processing applications. As well as acquiring technical skills, they experience the challenge of teamwork and defending their solution in front of the class.

Student feedback was consistent with the observations and code analysis made throughout the project, and show that the teaching goals were reached and that students absorbed the concepts taught in the course and plan to use them in the future.

ACKNOWLEDGMENT

The authors would like to thank the students of the academic year 2011/2012 and 2012/2013.This work was supported in part by the Research Council KU Leuven: GOA TENSE (GOA/11/007) and by the Flemish iMinds projects. In addition, this work was supported by the Flemish Government, FWO G.0550.12N, by the Hercules Foundation AKUL/11/19 and by the Intel University Program. Markus Ullrich is an FWO Doctoral researcher.

REFERENCES

[1] P. Schaumont, *A Practical Introduction to Hardware/Software Codesign*. Springer, 2010.
[2] ——, "A Senior-Level Course in Hardware/Software Codesign," *Education, IEEE Transactions on*, vol. 51, no. 3, pp. 306 –311, aug. 2008.
[3] R. H. Klenke, J. H. Tucker, and J. M. Blevins, "A new hardware/software codesign environment and senior capstone design project for computer engineering," *IEEE Microelectronic Systems Education (MSE03)*, pp. 66 – 67, June 2003.
[4] W. Wolf, "A decade of hardware/software codesign," *IEEE Transactions on Computer*, pp. 38 – 43, April 2003.
[5] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, pp. 120–126, 1978.
[6] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot, *Handbook of Applied Cryptography*, 1st ed. Boca Raton, FL, USA: CRC Press, Inc., 1996.
[7] P. L. Montgomery, "Modular Multiplication Without Trial Division," *Mathematics of Computation*, vol. 44, pp. 519–519, 1985.
[8] J. Wharton, "An Introduction to the Intel®MCS-51™Single-Chip Microcomputer Family," Intel Corporation., Application Note AP-69, 980 May.
[9] S. Dutta. (1995) SDCC - Small Device C compiler. [Online]. Available: http://sdcc.sourceforge.net
[10] M. Ošmera. (2011) MCU8051IDE. [Online]. Available: http://mcu8051ide.sourceforge.net/
[11] Ç. K. Koç, T. Acar, and B. S. Kaliski, Jr., "Analyzing and comparing montgomery multiplication algorithms," *IEEE Micro*, vol. 16, no. 3, pp. 26–33, 1996.
[12] Ç. K. Koç and T. Acar, "Montgomery Multiplication in GF(2k)," *Designs, Codes and Cryptography*, vol. 14, pp. 57–69, 1998.
[13] Xilinx, "Virtex-II Pro Data Sheets," Tech. Rep., 2011.
[14] L. Uhsadel, M. Ullrich, B. Preneel, and I. Verbauwhede, "Interface design for mapping a variety of RSA exponentiation algorithms on a HW/SW co-design platform," in *23rd IEEE Application-specific Systems, Architectures and Processors (ASAP 2012)*. IEEE, 2012.
[15] D. Chuda, P. Navrat, B. Kovacova, and P. Humay, "The issue of (software) plagiarism: A student view," *Education, IEEE Transactions on*, vol. 55, no. 1, pp. 22 –28, feb. 2012.

Leif Uhsadel is a Ph.D. candidate and teaching assistant with the COSIC group of the university of Leuven (KU Leuven). His research interests are efficient implementation of cryptographic algorithms on constrained devices and side-channel attacks. Uhsadel has a M.S. in IT security from the university of Bochum (Ruhr-Universität Bochum).